Blu-Ice and the *Distributed Control System*: software for data acquisition and instrument control at macromolecular crystallography beamlines

Timothy M. McPhillips,[a] Scott E. McPhillips,[a] Hsiu-Ju Chiu,[a]† Aina E. Cohen,[a] Ashley M. Deacon,[a] Paul J. Ellis,[a] Elspeth Garman,[b] Ana Gonzalez,[a] Nicholas K. Sauter,[a]‡ R. Paul Phizackerley,[a] S. Michael Soltis[a] and Peter Kuhn[a]*

[a]Stanford Synchrotron Radiation Laboratory, 2575 Sand Hill Road, MS 69, Menlo Park, CA 94025, USA, and [b]Laboratory of Molecular Biophysics, Rex Richards Building, South Parks Road, Oxford OX1 3QU, UK. E-mail: pkuhn@stanford.edu

The *Blu-Ice* and *Distributed Control System* (*DCS*) software packages were developed to provide unified control over the disparate hardware resources available at a macromolecular crystallography beamline. *Blu-Ice* is a user interface that provides scientific experimenters and beamline support staff with intuitive graphical tools for collecting diffraction data and configuring beamlines for experiments. *Blu-Ice* communicates with the hardware at a beamline *via DCS*, an instrument-control and data-acquisition package designed to integrate hardware resources in a highly heterogeneous networked computing environment. Together, *Blu-Ice* and *DCS* provide a flexible platform for increasing the ease of use, the level of automation and the remote accessibility of beamlines. *Blu-Ice* and *DCS* are currently installed on four Stanford Synchrotron Radiation Laboratory crystallographic beamlines and are being implemented at sister light sources.

Keywords: macromolecular crystallography; data acquisition; instrument control; graphical user interface; remote access; beamline automation.

1. Introduction

The increasing demand for macromolecular crystallography beamline capacity at synchrotron light sources requires that the utilization of these resources be maximized (Mitchell *et al.*, 1999). Current approaches to improving efficiency include the development of software for enabling turnkey operation (Skinner & Sweet, 1998), increasing automation (Abola *et al.*, 2000) and providing remote access (Chiu *et al.*, 2002). A significant hurdle to developing such software is the heterogeneity of the computing and instrument-control infrastructure associated with the beamline. For example, beamline 9-2 at the Stanford Synchrotron Radiation Laboratory (SSRL), which was the original test bed for *Blu-Ice* and the *Distributed Control System* (*DCS*), comprises over 50 motors associated with the beamline optics and experimental apparatus, numerous attenuation and elemental foils, three ion chambers, an X-ray fluorescence detector, a multichannel analyzer, and a fast-readout large-area CCD detector. Owing to distinct operating constraints of the individual instruments, the low-level software controlling this hardware is not executed on a single computer; rather, a total of six computers running five different operating systems are currently required at this beamline. Thus, a comprehensive software approach is needed to unify the control over hardware resources scattered across a heterogeneous network of computer systems. The *Blu-Ice* and *DCS* software packages solve this problem and provide an effective architecture for developing intuitive graphical user interfaces, automating beamlines and enabling remote beamline operation.

2. *Blu-Ice*

Blu-Ice is a graphical user interface that provides beamline experimenters and support staff with unified control over all instrumentation at a particular beamline. *Blu-Ice* allows researchers to efficiently configure a beamline for their experiment, determine optimal data-collection parameters and carry out the data collection itself. It also enables support staff to operate and configure all the software-controlled elements of a beamline. *Blu-Ice* hides the underlying complexity of the beamline instrumentation and places the interfaces to key automation features within a single intuitive graphical environment. *Blu-Ice* also provides collaborative and remote-access capabilities that allow multiple users to run their own instances (independent executions) of *Blu-Ice* to view a single beamline from either local or remote locations. Experimenters run multiple instances of *Blu-Ice* in order to work together over the network and control data-collection experiments from offsite. Support staff run *Blu-Ice* from local and home offices to provide assistance to users and troubleshoot problems remotely.

Blu-Ice runs within a single window on the workstation desktop (Fig. 1). Graphical widgets common to all *Blu-Ice* tasks are located in the menu bar at the top or in the status bar at the bottom of the *Blu-Ice* window. The status bar indicates the current status of the beamline including the energy of the X-ray beam, the state of the high-speed shutter and the activity of the detector. A scrolling log window displays a time-stamped record of all operations carried out on the beamline since that instance of *Blu-Ice* was started. Interfaces for carrying out distinct experimental or beamline control tasks are divided among the tabs of a notebook-style widget located between the menu bar and the log window. These specialized interfaces are described below.

2.1. Hutch tab

The Hutch tab provides the experimental user with views of the most relevant beamline hardware and allows the user to configure the beamline for an experiment (Fig. 1). The names of hardware devices and their current positions are arranged around still photographs, drawings or animated cartoons of the beam-conditioning system, the goniometer and the detector, so that users may easily associate the device names with the actual instruments they see in the experimental hutch. Drop-down menus for each moveable device provide suggested positions spanning the allowable range of motion. The user can specify the positions of the goniometer axes, the position of the detector, the distance between the sample and beam stop, the beam energy, the beam size, and the percent beam attenuation in this fashion. Entering new values for one or more device positions and pressing the 'Start' button begins the movement of the devices to the new positions.

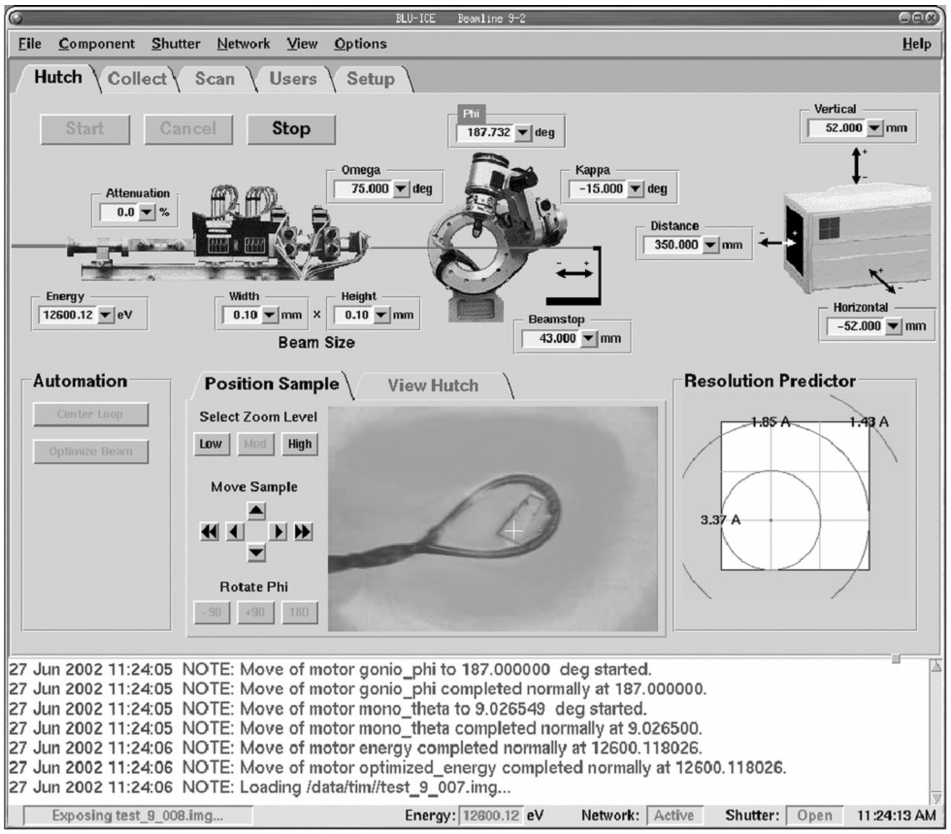
The Hutch tab indicates the current status of these devices on all instances of *Blu-Ice* associated with the particular beamline. The names of moving devices are highlighted and the positions of the

† Present address: Tularik Inc., Two Corporate Drive, South San Francisco, CA 94080, USA.
‡ Present address: Lawrence Berkeley National Laboratory, One Cyclotron Road, Mail Stop 4-230, Berkeley, CA 94720, USA.

moving devices are updated in real time. The X-ray beam is indicated graphically when the shutter is open.

The resolution predictor graphically indicates the position of the direct beam and the *d*-spacing of diffraction spots expected on different parts of the detector face. The resolution predictor updates dynamically as the detector position and beam energy devices change in value. This update also occurs as the user types trial values into the device-position windows, allowing the user to visualize the effect of changing these parameters without moving any motors. Conversely, the user may click on the resolution predictor to indicate the desired position of the direct beam with respect to the detector; new values for the horizontal and vertical detector positions are entered automatically into the corresponding entry boxes.

A small tabbed interface allows one of two video streams to be selected. The first video stream shows a live image of the sample on the goniometer and provides a click-to-center system; clicking on the image of the sample causes the clicked point to move to the rotation center of the goniometer. Buttons are provided for rotating the goniometer φ axis, for changing the sample camera-zoom level and for fine positioning of the sample. The second video stream presents a view from a pan-tilt-zoom camera mounted in the experimental hutch. Buttons are provided to move the camera to predefined orientations. These video streams greatly simplify the operation of the beamline from outside the hutch, allow the beamline to be operated safely from remote locations and enable remote observers to watch the motion of the instrumentation during an experiment.



```
27 Jun 2002 11:24:05  NOTE: Move of motor gonio_phi to 187.000000  deg started.
27 Jun 2002 11:24:05  NOTE: Move of motor gonio_phi completed normally at 187.000000.
27 Jun 2002 11:24:05  NOTE: Move of motor mono_theta to 9.026549  deg started.
27 Jun 2002 11:24:05  NOTE: Move of motor mono_theta completed normally at 9.026500.
27 Jun 2002 11:24:06  NOTE: Move of motor energy completed normally at 12600.118026.
27 Jun 2002 11:24:06  NOTE: Move of motor optimized_energy completed normally at 12600.118026.
27 Jun 2002 11:24:06  NOTE: Loading /data/tim//test_9_007.img...
Exposing test_9_008.img...      Energy: 12600.12 eV    Network: Active    Shutter: Open    11:24:13 AM
```

Figure 1
The *Blu-Ice* graphical user interface. The Hutch tab is shown.

2.2. Collect tab

The Collect tab is used to specify, control and monitor single-wavelength and multiwavelength diffraction data-collection experiments. At the right side of the window is a tabbed notebook containing up to 16 individual run definitions. A run is a sequence of exposures taken on a single sample with varying exposure settings. For each run, the user may specify parameters that apply to all images in the run including a prefix from which image file names will be generated, a directory in which to store the images, the detector readout mode, the detector distance and the axis of the κ goniometer to rotate during an exposure (either φ or ω). Additional parameters specify the sequence of images to be collected, including the starting and ending rotation axis positions and the corresponding frame numbers; the rotational angle spanned by each exposure; and the requested exposure time for each exposure. The user may specify up to five different energies at which to collect each frame of data. When using the φ axis, the user may request that the inverse-beam equivalent of each exposure (an exposure starting 180° away) be collected as well. A final parameter specifies the size of the wedge of data to be collected between energy changes and switches between the two halves of an inverse-beam experiment.

The central portion of the Collect tab provides widgets for controlling the progress of the experiment. Pressing the ‘Start’ button initiates data collection on the currently selected run tab. Pressing ‘Pause’ while data collection is in progress stops data collection after completion of the current image. Data collection may be resumed at the next frame of a paused run simply by pressing ‘Start’. A run-sequence preview window lists the exposures defined by the currently selected run in the order in which the system will collect them. The name of the image file, the starting φ or ω angle, and the energy corresponding to each image are displayed. The next image to be collected is highlighted. Double-clicking on a different image causes the system to jump to that point in the run, either skipping forward or backward in the sequence. The run sequence updates dynamically as the user modifies run definitions, providing rapid feedback about how the run-definition parameters affect the sequence. A dose-mode checkbox allows the user to indicate whether exposure times should be corrected dynamically for variations in beam intensity.

The left side of the Collect tab is occupied by a view of the last collected diffraction image. The user may adjust the centering, zoom level and contrast level of the image. Overloaded pixels are indicated graphically. At high zoom levels the pixel values are displayed.

2.3. Scan tab

The Scan tab allows the user to perform X-ray fluorescence scans of samples mounted on the goniometer. A two-tabbed notebook occupies the right side of the window. The first tab displays a periodic table that allows the user to select any of the X-ray absorption edges accessible at the beamline. Clicking on one of the edges automatically generates optimal experimental parameters for the scan. A fluorescence scan of the sample may then be performed by pressing the 'Start Scan' button.

Blu-Ice automatically runs a modified version of the program *Chooch* (Evans & Pettifer, 2001) on newly collected fluorescence data and displays the resulting f' and f'' curves. Based on these curves, *Blu-Ice* automatically selects three energies for a multi-wavelength anomalous diffraction (MAD) experiment: the inflection point of the absorption edge, the peak of the absorption edge and a remote energy (Hendrickson, 1991). These three energies are displayed below the plot and may be adjusted by dragging cursors on the plot. They may then be loaded into a run definition on the Collect tab with a single button click.

2.4. Setup tab

The Setup tab, accessible only to beamline support staff, provides a more comprehensive version of the Hutch tab. It presents an integrated view of all the instrumentation at the beamline and enables beamline scientists to efficiently commission, configure and troubleshoot the beamline. Cartoons of the beamline components allow the staff to manipulate beamline hardware elements such as monochromators, slits and goniometers without needing to remember numerous motor names, sign conventions and relative orientations of devices. Floating windows within the Setup tab allow multiple devices to be viewed. Motor-configuration parameters such as speeds, accelerations, scale factors and software limits, as well as other beamline hardware parameters, may be viewed or modified within the Setup tab. Both one- and two-dimensional scans of motor position *versus* ion-chamber readings may be performed. Finally, a command line allows all beamline control operations to be performed *via* typed commands or scripts written in the Tcl language. Such scripts greatly simplify troubleshooting of hardware.

3. *Distributed Control System* (*DCS*)

The capabilities of *Blu-Ice* depend heavily on the underlying *DCS* package. *DCS* is an instrument-control and data-acquisition system designed for highly heterogeneous networked computing environments. *DCS* uses a simple message-passing protocol for communication between low-level hardware control systems and graphical user interfaces. The *DCS* protocol is easy to port to any network-capable computing platform, including recent operating systems such as Linux and Microsoft Windows, legacy platforms such as OpenVMS, and even embedded systems with no operating-system support at all. Thus, *DCS* allows all hardware resources at a beamline to be placed under the control of a single control system, regardless of the computing platforms associated with those resources. Furthermore, *DCS* provides server-side scripting capability for implementing complex procedures spanning all of the hardware resources at a beamline. *DCS* is consequently a powerful platform for implementing sophisticated beamline automation features.

3.1. *DCS* architecture

DCS is characterized by a three-tier message-passing architecture (Fig. 2). The top tier of *DCS* consists of graphical user interfaces. Interactive tools for carrying out experiments and otherwise operating the beamline are implemented at this level. For the remainder of this paper, it will be assumed that the graphical user interface to *DCS* is *Blu-Ice* and that beamlines using *DCS* are dedicated to macromolecular crystallography. In principle, however, a wide variety of user interfaces could be developed for use with *DCS* on beamlines designed for other disciplines.

The bottom tier of the *DCS* architecture is composed of *DCS* hardware server (*DHS*) programs. A *DHS* program encapsulates low-level control of one or more physical devices such as motors, shutters, detectors *etc.* *DHS* programs are typically run on computers dedicated to hardware control. These computers are in turn located on a dedicated network isolated from the public network at the light source.

The central tier contains the *DCS* server, *DCSS*. One instance of *DCSS* runs at a particular beamline, routing requests from *Blu-Ice* to the appropriate hardware servers and broadcasting replies from the hardware servers back to *Blu-Ice*. *DCSS* is normally run on a multi-homed machine with network addresses on both the public network where *Blu-Ice* is run and the hardware-control network where *DHS* programs are executed.

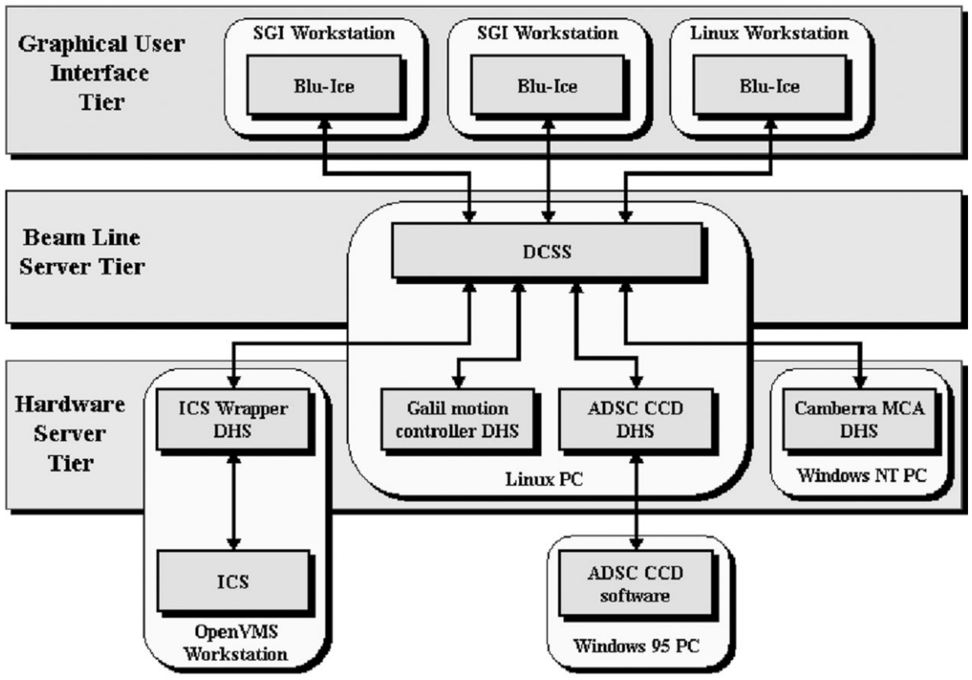


Figure 2
The three-tiered architecture of *DCS* as currently implemented at SSRL beamline 9-2. The instances of *Blu-Ice* shown are examples. Not all *DHS* programs are shown. Hardware supported natively include the ADSC Q4 and Q315 CCD systems; the Mar Research *mar*345 imaging-plate system; the Canberra DSA-2000 multichannel analyzer; and the Galil DMC-1000 and DMC-2100 series motion controllers. CAMAC modules supported *via* the *ICS* control system (George, 2000) include the DSP E500 motion controller, the DSP RTC018 real-time clock, the Kinetic Systems 3610 hex scalar, the Joerger D/A16 analog output module and the Joerger QOR quad output register.

3.2. *DCS* messaging protocol

All communication between the programs in the *DCS* architecture uses a message-passing protocol that runs on top of TCP/IP. Each *DCS* message consists of a variable-length text payload, a variable-length binary payload and a fixed-length ASCII header that lists the lengths of the payload sections. The binary payload is currently used only for passing authentication information between *Blu-Ice* and *DCSS*. All other communication in *DCS* uses the text portion of the *DCS* messages exclusively.

The text payload of each *DCS* message consists of a string composed of white-space-separated tokens. The first token in each message is the command; all remaining tokens are arguments to the command. For example, the following is a command from *Blu-Ice* to *DCSS* requesting that the motor *table_vert_1* be moved to the absolute position of 10.0 mm:

gtos_start_motor_move table_vert_1 10.0.

The *DCS* protocol is completely asynchronous (*i.e.* not strict client/server). This asynchronous message-passing architecture allows for simultaneous operation of any number of different devices at the beamline. Any number of different motor moves may be started independently without having to wait for previous moves to complete and without needing to define aggregates of devices that move together. For example, users of *Blu-Ice* may start to move the detector positioner and then begin centering a crystal on the motorized sample stage while the detector is still moving. Similarly, any number of distinct automation tasks may be performed simultaneously.

The *DCS* protocol eliminates the need for polling devices from within *Blu-Ice*. A *DCS* hardware server is responsible for reporting autonomously when a motor motion or other operation is complete. An instance of *Blu-Ice* need only wait for the completion message to arrive. Similarly, hardware servers provide updates of motor positions several times a second whenever motors are moving. All instances of *Blu-Ice* are able to display the current positions of all motors simply by processing all incoming motor-status messages as they arrive and updating the graphical user interfaces accordingly.

3.3. Distributed hardware servers (*DHS*)

Any program that supports the *DCS* messaging protocol can act as a hardware server in a beamline controlled by *DCS*. A *DHS* program works as a protocol translator. The generic *DCS* messages are translated by a *DHS* into the control protocol specific to a particular type of device. This allows *Blu-Ice*, *DCSS* and the *DCS* protocol to remain entirely generic. All hardware-specific code is encapsulated within the hardware server dedicated to a particular device. A typical beamline will run several *DHS* programs on different computers, and those computers may run different operating systems. This allows operating-system-specific device libraries to be exploited by a *DHS*. A *DHS* may also wrap another control system. For example, more than half of the motors at each SSRL beamline are controlled at a low level by the standard SSRL control system, *ICS* (George, 2000). A *DHS* is used to wrap *ICS* so that these devices may be controlled *via* the *DCS* protocol. This type of *DHS* allows *Blu-Ice* to seamlessly integrate multiple low-level control systems at the same beamline. Furthermore, this approach allows beamline developers to deploy *Blu-Ice* rapidly on beamlines with different hardware systems than those used at SSRL, as long as some basic control system is already in place at those beamlines.

3.4. DCS server (*DCSS*)

The *DCS* server is the central component of *DCS* and provides a number of critical services. First, *DCSS* stores the state of the beamline, including the positions of all motors. Second, *DCSS* is a message router, facilitating controlled communication between instances of *Blu-Ice* and the hardware servers. *DCSS* examines incoming messages from *Blu-Ice* and forwards messages that refer to particular devices to the appropriate hardware servers. Conversely, responses from hardware servers are forwarded by *DCSS* to all instances of *Blu-Ice*. In this way, *DCSS* provides all graphical user interfaces with the same status information, allowing them to remain synchronized at all times.

By controlling the flow of instructions to the hardware servers, *DCSS* prevents damage to the beamline hardware due to conflicting instructions from different instances of *Blu-Ice*. *DCSS* allows only one *Blu-Ice* instance to control the beamline hardware at one time. This instance is referred to as the *active client*. All other instances of *Blu-Ice*, the *passive clients*, may monitor all beamline operations and use passive features such as the resolution predictor, but they cannot issue commands to control the beamline without first becoming the active client. Thus, two users of *Blu-Ice* can never inadvertently request the control system to perform contradictory operations.

DCSS also enforces authentication of users of *Blu-Ice* and uses an access-control list to determine which users are allowed to perform what operations. A connecting *Blu-Ice* instance must respond correctly to a challenge sent by *DCSS* before being allowed to access the beamline. At SSRL, the challenge–response sequence is based on a per-user private key known to both *DCSS* and *Blu-Ice*. This allows users of *Blu-Ice* to authenticate automatically without typing in a password. The access-control list restricts different users to distinct levels of access to the beamline hardware. At the lowest level of access, a *Blu-Ice* user is able to monitor operations at the beamline but cannot become the active client or affect the beamline in any way. At the highest level, all devices can be operated and configured.

These features of *DCSS* greatly simplify the implementation of *DHS* programs. A designer of a *DHS* program does not need to take into account the existence of multiple user interfaces, active and passive clients, security and privilege issues, or the nature and activity of other *DHS* programs at the same beamline. Each *DHS* must simply respond to a single coherent stream of instructions that are guaranteed to be safe to carry out. Similarly, *DCSS* hides the nature and location of the various *DCS* hardware servers from the user-interface tier, allowing *Blu-Ice* to view and make use of a uniform pool of hardware resources at the beamline.

3.5. Scripting engine

A fully automated, efficient and optimized MAD experiment requires the sequenced and synchronized operation of a large number of devices. Differences between beamlines require that the procedures that control diffraction data collection, automatic beam optimization, fluorescence scans *etc.* be easy to design, understand, customize and test. *DCS* provides a powerful server-side scripting interface, the *scripting engine*, for implementing such beamline automation procedures.

The scripting engine is implemented as a Tcl interpreter executing within the process context of *DCSS*. Locating the interpreter at this level allows scripts to access all hardware resources at a beamline uniformly. It also allows these procedures to execute independently of any graphical user interface. Users may close all copies of *Blu-Ice* after starting a data collection and the scripting engine will complete the entire experiment.

Two types of scripts may be defined and executed within the scripting engine: *scripted devices* and *scripted operations*. Both types of scripts are written in the Tcl scripting language and may be added to the system or modified without recompiling any part of *DCS*. A scripted device is a virtual motor that may be composed of any number of real motors or other virtual motors. A scripted device may be moved like a real motor from *Blu-Ice* or from other scripts. Simple examples of scripted devices on SSRL beamlines are the *table_vert* and *table_pitch* devices, both of which move the real motors *table_vert_1* and *table_vert_2* according to equations included within the scripts. Scripted devices may be nested indefinitely.

A scripted operation is a more general construct that allows beamline automation procedures to be implemented. For example, the procedure to automatically maximize the intensity of the beam by adjusting the table position on an SSRL beamline is implemented as a scripted operation. As such, it can be initiated from *Blu-Ice* or from within other scripted operations and scripted devices. In particular, the scripted operation that performs data collection also executes this beam-optimization script periodically to compensate for beam motion that may occur.

Multiple scripted devices and operations may run simultaneously, with scripts automatically and safely yielding control to each other when waiting for asynchronous events such as completion of motor moves or other operations. Concurrency in the scripting engine is based on an event model rather than a more complex multi-threading model (Ousterhout, 1998). *DCS* script writers may largely ignore the complexities associated with thread-based concurrent programming, such as the need to protect shared data and avoid deadlock. Consequently, beamline scientists with minimal software-development experience may describe the behavior of the beamline in simple linear scripts that are easy to write, debug and understand. Example scripts that illustrate how differences between beamlines can be managed using scripted devices and operations may be found in the online *DCS* administrator's manual (McPhillips, 2002*a*).

4. Software portability and availability

The large number of operating systems already in use at synchrotron radiation beamlines, combined with the new computing platforms likely to be exploited by future hardware vendors, necessitates a truly cross-platform approach to beamline software development. Furthermore, there are significant performance constraints on many of the software subsystems within *Blu-Ice*/*DCS*. High-performance multi-threaded compiled code is required when the overriding goal is to collect as many diffraction images as quickly as possible. Therefore, the development strategy for *DCS* and *Blu-Ice* was to use existing cross-platform tools such as Tcl/Tk for graphical user interface development and scripting but to use multi-threaded C and C++ programs for high-performance server software. The simple text-based *DCS* communication protocol was developed to minimize operating-system dependencies related to network communication as described above, and a custom library (XOS) was written in C to provide cross-platform compatibility wherever compiled C/C++ code was required.

4.1. Cross-operating-system library (XOS)

The XOS library provides a portable application programming interface (API) for system calls that are otherwise highly platform dependent. Designed specifically for multi-threaded distributed applications, the library supports threads, mutexes, semaphores, message queues, Win32-style messages (even on non-Windows platforms), memory mapped files and TCP/IP network communication. The library is written in C and uses the preprocessor to compile different code depending on the target platform so that there is no run-time overhead.

XOS currently allows the programmer to write high-performance code that runs on Linux, Irix, Tru64 Unix, OpenVMS and Win32 platforms without modification. Support for additional operating systems is easy to add, since code common to different platforms is shared. For example, the XOS code supporting threads, mutexes and semaphores is shared between the Unix-like platforms and OpenVMS, since these platforms support the POSIX threads API, whereas the underlying XOS code supporting threads on Win32 is distinct. The XOS API for managing threads is, nevertheless, identical on every platform. Not only does the XOS library hide system differences from the application programmer, it also simplifies the function calls and hides the complexities of the particular operating system APIs.

4.2. Graphical user interface development using Tcl/Tk

Although the high performance of compiled C/C++ is required for *DCSS* and some *DHS* programs, Tcl/Tk provides a rapid-development environment for implementing graphical user interfaces (Ousterhout, 1994). In addition, Tcl/Tk achieves a very high level of cross-platform compatibility, since a single distribution of source code underlies all Tcl/Tk distributions running on Unix-like platforms, 32-bit Windows and MacOS. *Blu-Ice* is written using the [Incr Tcl/Tk] extension to Tcl/Tk, which adds true object-oriented programming features to Tcl (McLennan, 1995).

4.3. Availability

Blu-Ice/*DCS* is open source software distributed under an MIT-style free software license (McPhillips, 2002*b*). Online access to the *Blu-Ice*/*DCS* software distribution at SSRL has been provided to collaborators at the Advanced Light Source (California), the Advanced Photon Source (Illinois), Brookhaven National Laboratory (New York), the Synchrotron Radiation Research Center (Taiwan) and ELETTRA (Italy). The Concurrent Versions System (Cederqvist, 1992) is used for version control, software distribution, and incorporation of software components and patches contributed by the collaborators. Bugzilla (Mozilla Organization, 2002) is used for managing feature requests and bug reports. The *Blu-Ice*/*DCS* software distribution includes extensive online documentation of installation and administration procedures. Additional information about the software distribution may be found at the *Blu-Ice*/*DCS* distribution web site. (McPhillips, 2002*b*).

The authors would like to thank the entire Structural Molecular Biology group at SSRL for their support and efforts with this project. In addition, we would like to thank the many external users of the macromolecular beamlines at SSRL who all gave us invaluable feedback on the functionality of the software. This project was carried out at SSRL and supported by the US National Institutes of Health, National Center for Research Resource, Biomedical Technology Program (P41-RR-01209), the National Institutes of General Medical Sciences, and the Department of Energy, Office of Biological and Environmental Research. SSRL is operated by Stanford University on behalf of the US Department of Energy, Office of Basic Energy Sciences.

References

Abola, E., Kuhn, P., Earnest, T. & Stevens, R. C. (2000). *Nat. Struct. Biol.* **7**, 973–977.

Cederqvist, P. (1992). *Version Management with CVS*, http://www.cvshome.org/docs/manual/.

Chiu, H.-J., McPhillips, T., McPhillips, S., Sharp, K., Eriksson, T., Sauter, N., Soltis, M. & Kuhn, P. (2002). *Networked Learning*, pp. 128–134. NAISO Academic Press.

Evans, G. & Pettifer, R. F. (2001). *J. Appl. Cryst.* **34**, 82–86.

George, M. J. (2000). *J. Synchrotron Rad.* **7**, 283–286.

Hendrickson, W. A. (1991). *Science*, **254**, 51–58.

McLennan, M. J. (1995). *Proceedings of the USENIX Third Annual Tcl/Tk Workshop*, Toronto, Ontario, Canada, July 1995. Berkeley, CA: USENIX Association. (http://www.usenix.org/publications/library/proceedings/tcl95/mclennan.html.)

McPhillips, S. E. (2002*a*). *Blu-Ice/DCS Administrator's Manual*, http://smb.slac.stanford.edu/blu-ice/dcsAdmin/.

McPhillips, S. E. (2002*b*). *The Blu-Ice/DCS Software Distribution*, http://smb.slac.stanford.edu/blu-ice/.

Mitchell, E., Kuhn, P. & Garman, E. (1999). *Structure*, **7**, 111–121.

Mozilla Organization (2002). *mozilla.org*, http://www.mozilla.org/.

Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit.* Reading, MA: Addison Wesley.

Ousterhout, J. K. (1998). *IEEE Comput.* **31**, 23–20.

Skinner, J. M. & Sweet, R. M. (1998). *Acta Cryst.* D**54**, 718–725.

Received 5 July 2002 • Accepted 22 August 2002